The Globalization of Manufacturing in the Digital Communications Era of the 21st Century: Innovation, Agility and the Virtual Enterprise

Proceedings of the Tenth International IFIP WG 5.2/5.3 Conference PROLAMAT 98

83
A comparison of different HCI styles in a KBSE system

S. G. Lazzeri
RECOM Technologies
NASA Ames Research Center, Mail-Stop 269-2,
Moffet Field, CA 94035-1000, USA, Phone: (650)-604-1105,
FAX: (650)-604-3594, email: olmeca@ptolemy.arc.nasa.gov

Abstract
There has been a sizable amount of research about the relative merits of different styles of human computer interaction and information presentation. However, it has not been possible to declare a definitive winner between graphical and text based interfaces. This paper compares graphical and text based interfaces for formal specifications in the context of Amphion, a generic knowledge based software engineering system, which has been applied to the domain of solar system kinematics. Preliminary observations favor the text based interface due to its ability to present details, but the graphical interface has also advantages, such as the compactness of its displays. The promising alternative of using a hybrid interface and potential applications domains for Amphion, such as manufacturing and education are discussed.

Keywords
human computer interaction, text based, graphical, visualization, KBSE, HCI knowledge based, software engineering, formal specifications, client server

1 INTRODUCTION

Every interaction style provides particular advantages that may be more relevant than those provided by other styles depending on the domain and characteristics of the user population in which they are applied. This is supported by a vast body of research in human computer interaction (HCI).

For example, Ladret (1991) uses text and graphics to represent Prolog programs, Lazzeri (1996) studies visual and textual modes of advice in a chess learning environment, and Salton (1994) proposes different hypertext structures to organize large text files. This paper compares text based and graphical interfaces for formal specifications in the context of the Amphion System (e.g. Lowry, 1994). It focuses in those features that make the text based interface desirable as a substitute or a complement to a graphical interface.

The remainder of this paper is organized as follows: Section 2 describes the Amphion System, Section 3 describes Amphion's graphical interface, Section 4 discusses the motivation for developing alternative interfaces, Section 5 describes the text based interface for Amphion, Section 6 presents a comparison between both interfaces, and Section 7 presents the conclusions and future work.

2 THE AMPHION SYSTEM

Amphion is a generic Knowledge Based Software Engineering (KBSE) system that targets scientific subroutine libraries. It implements a formal approach to domain oriented software design environments, based on declarative domain theories, formal specifications, and deductive program synthesis. A declarative domain theory defines the semantics of a domain oriented specification language and its relationships to implementation level subroutines. Deductive program synthesis ensures that specifications are correctly implemented. Amphion has been successfully applied to the domain of solar system kinematics. For this specific implementation, the subroutines targeted are those of the NAIF libraries developed at JPL (e.g. Rugaber, 1996).

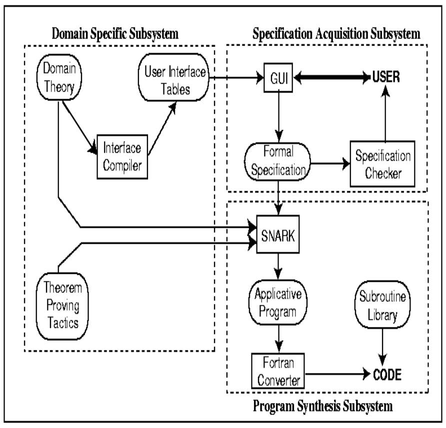
2.1 The original Amphion version

Figure 1 shows the architecture for Amphion's original version. That version, written specifically for Unix based platforms, was organized into three main components.

The Domain Specific Subsystem contained all the domain specific knowledge and interface objects that were necessary for the system to function. It was necessary both to guide the user during the creation of a specification, and to provide the Program Synthesis Subsystem with the necessary axioms to successfully create the FORTRAN program from the original specification.

The Specification Acquisition Subsystem allowed users to develop and reuse formal specifications through a graphical user interface (GUI). This component also included a specification checker that made sure the specifications were correct.

The Program Synthesis Subsystem was in charge of converting the user specification into a FORTRAN program composed of subroutine calls to the NAIF libraries. SNARK (e.g. Stickel, 1996) is a theorem prover used to translate the user's specification into a logical program, which was then translated into the final FORTRAN program. The resulting program was made available to the user who could then compile and run it to produce either printed results or animations to satisfy his/her requests.



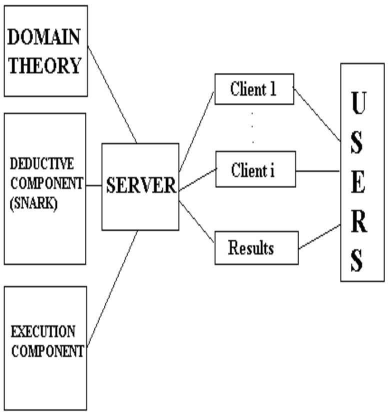


Figure 1 Original Amphion's architecture   Figure 2 Amphion's architecture.

The main limitations of the original Amphion version were:

- It was only accessible from Unix based platforms.
- It handled only one user per Amphion process at a time.
- It was limited to one specific user interface.
- It required manual compilation/execution of the Amphion generated program.
- Domain theory development was primarily done by encoding the knowledge directly as programming language (i.e. LISP) code.

2.2 Client Server Approach

Amphion's capabilities have already been useful to NASA space scientists. It is now desired to extend these capabilities to a wider audience. The current challenge is to make these capabilities available to different domains and also for educational purposes at different levels. In order to achieve this goal, a www based client server development environment has been built. The objective of this environment is to allow as wide an audience as possible to take advantage of Amphion's functionality. Figure 2 presents a diagram describing the architecture of the new environment for the Amphion system. In that figure, several components can be identified.

The main improvement over the previous architecture is the addition of the Server component, which acts as middleware, facilitating the interaction between a number of clients and the different components of the Amphion system. Each client encapsulates a simplified version of the Specification Acquisition Subsystem as described in the original Amphion architecture. The Server can handle several independent sessions at the same time, thus providing service to different users simultaneously.

The Domain Specific Subsystem (e.g. domain theory) and the Program Synthesis Subsystem (e.g. deductive component) remain mostly unchanged. However, a new component, the execution component, now takes charge of automatically compiling and running the program. Once the deductive component has translated the user's specification into a software program, the server turns the source code to the execution component, which takes care of compiling and linking the program to the appropriate libraries. The result of this process is an application program that is displayed to the user.

The interaction with the user is entirely handled through the clients and application programs generated by the system. Therefore, the user can specify problems and obtain results without having any knowledge of the low level details of the NAIF libraries or FORTRAN programming. Currently there are two different types of clients, one of them is graphical and the other is text based. They both are written as Java applets so they can be invoked using a web browser. However, since there is a well established protocol for communication between the server and the clients, it would be relatively easy to develop other types of clients that may be useful in different domains. The communication between server and clients is carried out through TCP IP sockets, so it is also possible to create stand alone clients that do not require a web browser to run.

The application programs generated by the system are run in the server side and displayed in the client machine using an X window, so it is necessary for the user to have an X Windows Server on the client machine to see its results. Ideally, it should be possible to develop clients with the specific purpose of displaying results, to eliminate the need to have an X Windows Server in the client. One such client is currently under development.

3 AMPHION'S GRAPHICAL INTERFACE

Amphion's original version allowed users to develop and reuse formal specifications through a graphical interface, which used graphical objects to represent the different components of formal specifications, and arrows to represent the relationships between components. This interface is particularly effective for the creation of specifications. Figure 3 shows an example of a specification as it is displayed by the graphical interface.

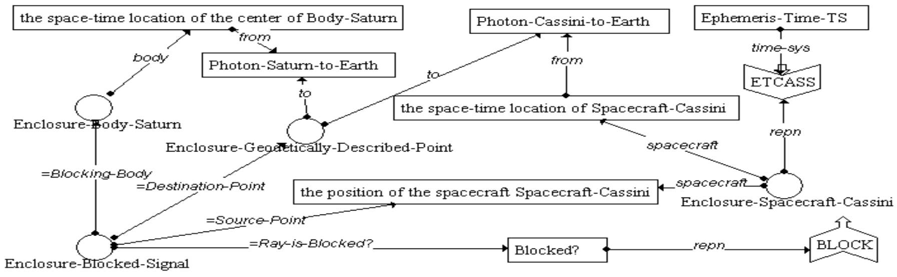


Figure 3 Program Cassini Visibility

This is a very good example to showcase the advantages of using a graphical representation for formal specifications. In this example all the components of the specification and their relationships can be clearly displayed. The circles are high level components, called enclosures, which can be expanded to display more detail. The rectangles are variables. The arrows represent relations between the different objects, and the irregular polygons represent input and output representations of variables. The output of this program is "BLOCK", which tells whether or not a signal sent from the spacecraft Cassini to Earth at the input time "ETCASS" is blocked by the planet Saturn. The high level semantics of this specification can be understood from Figure 3 even without a deep knowledge of the interface itself.

Unfortunately, not all specifications can be captured with the same simplicity as the Program Cassini visibility. The expanded version of the enclosure "Enclosure-Blocked-Signal" in the same program presents the common problem of having to present too many objects within a very limited space. As we can see in Figure 4, it is much more difficult to understand the semantics of this enclosure than it was to do it for the Program Cassini Visibility. The next section describes some of the main problems we can find when using graphical representations for formal specifications.

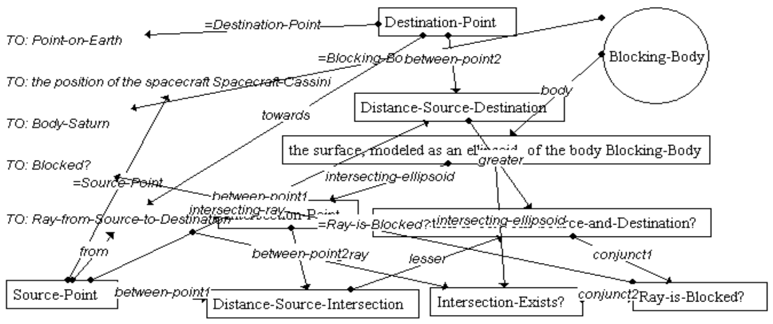


Figure 4 Enclosure Blocked Signal

4 MOTIVATION TO DEVELOP ALTERNATIVE INTERFACES

While the graphical interface handles small specifications well, there are a number of problems that increase with the size of the specifications used, such as:

1. The difficulty to visualize specifications that contain a large number of relationships between components because of the well known "spaghetti ball" phenomenon. This problem is found almost every time a complex structure is described graphically. For example, Ladret (1991) describes how it affects the graphical representation of Prolog programs. Figure 4 presents a typical example in the context of Amphion with so many intermingled arrows and objects that it is hard to tell where each arrow starts and ends.

2. The lack of standard graphical representations for objects in some domains makes the graphical representations less than intuitive in some cases. Since the objective of Amphion is to be a domain independent tool and there is no universal graphical notation for all domains, the problem of designing the icons to be used has to be solved in a case by case basis.
3. It is hard to see some of the details of the enclosures and its relationships from the graphical representations. In Figure 3 for example, it is clear that the enclosure "Enclosure-Blocked-Signal" has four links to elements that lay outside of itself, but it is not clear what specific elements inside this enclosure are connected to what external elements.
4. It is hard to hide unnecessary specification details. In Figure 4, it would be to our advantage to remove some of the detail. However, it is difficult to do this in a graphical representation without removing relevant information.

5 AMPHION'S TEXT BASED INTERFACE

In order to address these problems, a new, text based interface has been developed as an alternative/complement to the existing graphical interface. The text based interface avoids problems 1 (i.e. spaghetti ball) and 2 (i.e. lack of standard graphical representations) by representing the information about objects and their relationships in the form of natural language sentences. This representation makes it easy for novice users to better understand the specifications. By presenting the specification information organized in the form of outlines that can be selectively expanded and collapsed, the user is given a high degree of control over the information that is displayed at any given time. This capability minimizes problems 3 (e.g. selective information display) and 4 (e.g. selective information hiding) described before.

The outline for each formal specification classifies its components into different categories, such as inputs, outputs, constants, variables, and enclosures. More detailed information about any component in each of the categories can be obtained by expanding the components' relationships. For example, it is possible to expand the arguments used to compute a given component 'c', the components that use 'c' as an argument, the components that are equated to 'c', or the enclosure 'e' that contains 'c'.

The repeated application of this expanding ability allows the user to traverse the graph that represents the formal specification in any way. For example, it is possible to trace the outputs of a given program all the way back to the inputs that are required for its computation, going through all the intermediate values that need to be computed in order to obtain the outputs. It is also possible to start from the inputs and navigate towards the outputs by following the relationships that show how the inputs are used to compute new values, and how these new values are used to compute other new values until the outputs are produced. In general, the user should be able to navigate from any component to any other component in the specification by following the different relationships between components, assuming the specification can be represented by a connected graph.

Besides providing the ability to freely navigate through the specification, the text based interface can also provide specific views that correspond to particularly relevant traversals of the abstract graph that represents a specification. For example, the output to input traversal described before can be generated automatically by the interface. Information about each component can be provided at different levels, too. For example, it may include only the name of the component, the function used to compute it, or the complete description of the component, including its arguments. The complete descriptions can be either expanded in place, or displayed in a separate auxiliary window.

Figure 5 presents the text based display for the example Program-Cassini-Visibility shown also in Figure 3 for the graphical interface. Each element is represented by its name, which can be expanded to the full description of the element. There are six symbols to the left of each element. These symbols stand for different relations that can be requested from each element. The meaning of each symbol may be slightly different depending on the type of element to which they apply (e.g. either enclosure or simple element). Table 1 summarizes the meaning of each symbol for each case.

TABLE 1 Meaning of expanding symbols for text based interface

Symbol \ Type	*Enclosure*	*Simple Element*
@	Display Enclosure Outline	Display Arguments
=	Unused	Display Equalities
^	Unused	Display Parents
$	Display Enclosing Enclosure	Display Enclosing Enclosure
&	Display Interface	Display Hooks
+	Toggle Name/Description	Toggle Name/Description

Enclosure Outline is the high level outline defined for each enclosure. An outline classifies the relevant elements of an enclosure into meaningful categories, namely outputs, inputs, interface, constants, variables, photons, and enclosures.

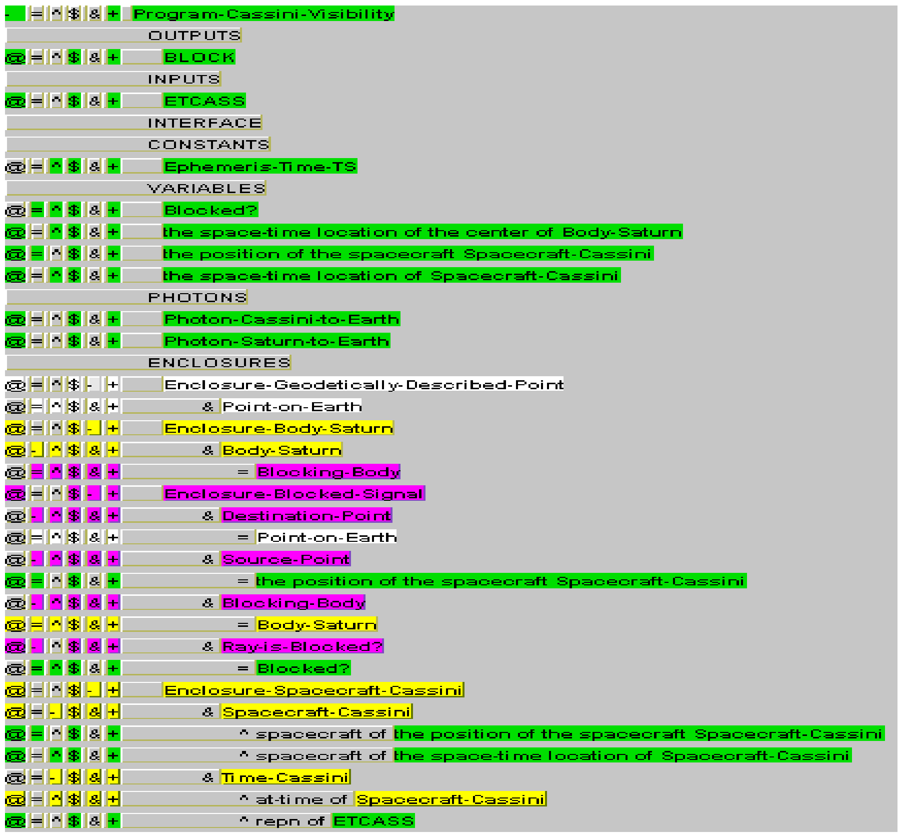
The arguments of a simple element ‘e’ are those simple elements whose values are required in order to compute ‘e’. The equalities of an element ‘e’ are those simple elements ‘e1’ for which there is an explicit equality relation between ‘e’ and ‘e1’. The parents of a simple element ‘e’ are those simple elements ‘s’ that require the value of ‘e’ in order to compute their own value.

The Enclosing Enclosure of a given element ‘e’ (e.g. either an enclosure or a simple element) is the enclosure that directly contains ‘e’. For example, the enclosure Enclosure-Blocked-Signal is the Enclosing Enclosure for element

'Destination-Point', and the enclosure 'Program-Cassini-Visibility' encloses the enclosure 'Enclosure-Blocked-Signal'.

The Interface of an enclosure 'E' refers to those simple elements 's' directly enclosed by 'E' that have at least one direct relationship (e.g. argument, parent, equality) to an element that is not enclosed by 'E' in any way. The Hooks of a simple element 's' are those elements 'h' which have a direct relationship to 's', but which lay in a different enclosure than 's'. The '+' symbol just toggles the id of an element 'e' between its name and its full description.

When any of the symbols described in Table 1 has been selected to expand the corresponding relation, it is turned into a '-' symbol to indicate that it is already expanded and cannot be expanded again.



```
- = ^ $ & +  Program-Cassini-Visibility
                 OUTPUTS
@ = ^ $ & +      BLOCK
                 INPUTS
@ = ^ $ & +      ETCASS
                 INTERFACE
                 CONSTANTS
@ = ^ $ & +      Ephemeris-Time-TS
                 VARIABLES
@ = ^ $ & +      Blocked?
@ = ^ $ & +      the space-time location of the center of Body-Saturn
@ = ^ $ & +      the position of the spacecraft Spacecraft-Cassini
@ = ^ $ & +      the space-time location of Spacecraft-Cassini
                 PHOTONS
@ = ^ $ & +      Photon-Cassini-to-Earth
@ = ^ $ & +      Photon-Saturn-to-Earth
                 ENCLOSURES
@ = ^ $ - +      Enclosure-Geodetically-Described-Point
@ = ^ $ & +            & Point-on-Earth
@ = ^ $ - +      Enclosure-Body-Saturn
@ - ^ $ & +            & Body-Saturn
@ = ^ $ & +                 = Blocking-Body
@ = ^ $ - +      Enclosure-Blocked-Signal
@ - ^ $ & +            & Destination-Point
@ = ^ $ & +                 = Point-on-Earth
@ - ^ $ & +            & Source-Point
@ = ^ $ & +                 = the position of the spacecraft Spacecraft-Cassini
@ - ^ $ & +            & Blocking-Body
@ = ^ $ & +                 = Body-Saturn
@ - ^ $ & +            & Ray-is-Blocked?
@ = ^ $ & +                 = Blocked?
@ = ^ $ - +      Enclosure-Spacecraft-Cassini
@ = - $ & +            & Spacecraft-Cassini
@ = ^ $ & +                 ^ spacecraft of the position of the spacecraft Spacecraft-Cassini
@ = ^ $ & +                 ^ spacecraft of the space-time location of Spacecraft-Cassini
@ = - $ & +            & Time-Cassini
@ = ^ $ & +                 ^ at-time of Spacecraft-Cassini
@ = ^ $ & +                 ^ repn of ETCASS
```

Figure 5 Program Cassini Visibility in Text Based Interface

Figure 5 also illustrates the use of color coding in order to identify the different enclosures, including the main program, in a specification. In this case the main program 'Program-Cassini-Visibility' is coded in green as well as every simple element that is directly enclosed by it. Similarly, 'Enclosure-Geodetically-Described-Point' and its elements are coded in white, 'Enclosure-Body-Saturn' in yellow, 'Enclosure-Blocked-Signal' in magenta, and 'Enclosure-Spacecraft-Cassini' in dark yellow. The expanding symbols are colored in the same way as the element they refer to, except in the case of symbols which are disabled (e.g. grayed). A grayed symbol means that no relations of the type specified by that symbol are available for its corresponding element. In a gray scale version of this paper, the coding is not so clear, but the enclosures should still be seen in different shades of gray.

To conclude this section, let us consider some of the potential applications of the text based interface. Even though the text based interface has been explicitly designed for Amphion structures, it provides a general framework that could be used to visualize and navigate through different structures that involve different types of components and relationships between those components. Structures that could possibly be traversed by this interface include finite state diagrams, Petri nets, and the organized results of a search engine. The approach used for the text based interface can be seen as an extension of the outline and collapser/expander programs commonly used to hierarchically browse file directories, with the particularity that this text based interface can present and combine different hierarchies over the same collection of elements.

6 INTERFACE COMPARISON

From figures 3 and 5, we can see that the text based interface provides more detailed information than the graphical interface at the expense of a somewhat lengthier representation. However, the ability of the text-based interface to expand and collapse different parts of the representation makes the trade off favor the text based interface.

For example, from figure 5 we can tell that the element 'Body-Saturn' in 'Enclosure-Body-Saturn' equals the element 'Blocking body' in 'Enclosure-Blocked-signal', and that the element 'Spacecraft-Cassini' in 'Enclosure-Spacecraft-Cassini' has parents (e.g. is an argument to) 'the position of the spacecraft Spacecraft-Cassini' and 'the space-time location of Spacecraft-Cassini' in the main program 'Program-Cassini-Visibility'.

On the other hand, we can at best get incomplete information from the graphical representation. All we know is that there is a Spacecraft argument coming out of 'Enclosure-Spacecraft-Cassini', but we do not know which one (i.e. there could be several Spacecraft definitions inside the enclosure). Similarly, we know there is an equality between an element inside 'Enclosure-Body-Saturn' and another element inside 'Enclosure-Blocked-Signal', but we do not know which ones. Even though it could be possible to add all these details to the graphical representation, it would hinder its compactness and simplicity, which are its best assets. Furthermore, this additional information may produce other readability problems as those seen in Figure 4.

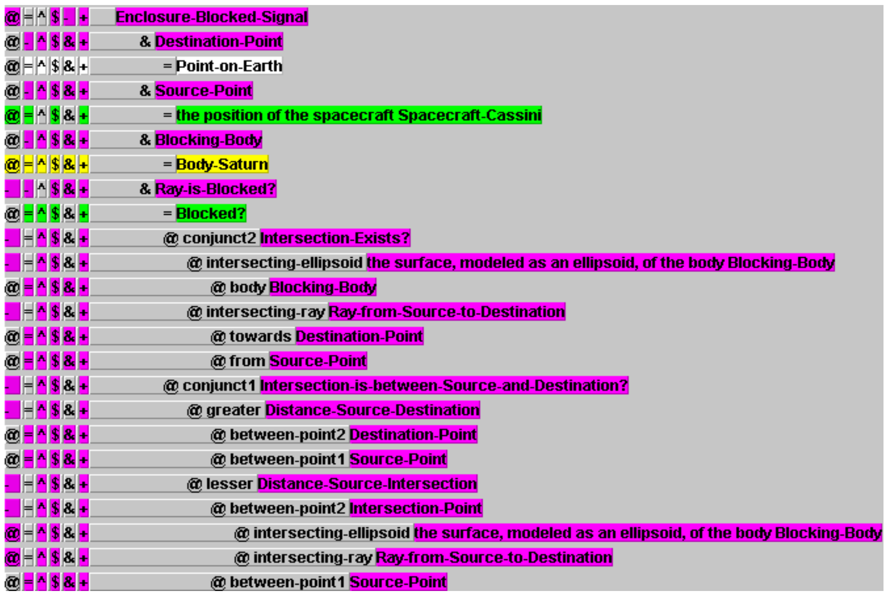
Figure 6 shows the text based representation for the enclosure presented in Figure 4 for the graphical interface. Here the readability advantages provided by the text based interface are more clearly observed. The interface for this enclosure consists of four simple elements, 'Destination-Point', 'Source-Point', 'Blocking-Body', and

'Ray-is-Blocked?'. Since the first 3 of these elements do not have any arguments, it is safe to assume that they all get their values from their external hooks, respectively 'Point-on-Earth', 'the position of the spacecraft Spacecraft-Cassini', and 'Body-Saturn'.

Since 'Ray-is-Blocked?' does have arguments, it is safe to assume that, unlike the other members of the interface, its value is used as an output to provide the value for its hook 'Blocked?'. All of this information can be easily extracted from this representation. Furthermore, we can easily see that 'Ray-is-Blocked?' is true if there is an intersection between the surface of the blocking body (e.g. Saturn) and the ray from the source (e.g. Cassini) to the destination (e.g. Earth), and this intersection lies between source (e.g. Cassini) and destination (e.g. Earth).

This example shows that the text-based interface can provide several distinct advantages over the graphical interface, such as better readability and easier access to details. The next example presents a comparison using a simpler example, which shows how the graphical interface has its virtues too.

Let us now look at a simpler specification used to compute the distance between Earth and Mars. This specification does not contain enclosures so it is more amenable to be displayed with the graphical interface. Figures 7 and 8 present the two alternative views. As we can see from these figures, both interfaces can display very clearly the details of the specification. The lack of enclosures avoids many of the complications encountered in the previous example. Since the graphical interface gives a somewhat more compact representation, it would perhaps be more appropriate for the display of this type of simpler specifications.



```
@ = ^ $ - +   Enclosure-Blocked-Signal
@ - ^ $ & +     & Destination-Point
@ = ^ $ & +       = Point-on-Earth
@ - ^ $ & +     & Source-Point
@ = ^ $ & +       = the position of the spacecraft Spacecraft-Cassini
@ - ^ $ & +     & Blocking-Body
@ = ^ $ & +       = Body-Saturn
- - ^ $ & +     & Ray-is-Blocked?
@ = ^ $ & +       = Blocked?
- = ^ $ & +       @ conjunct2 Intersection-Exists?
- = ^ $ & +         @ intersecting-ellipsoid the surface, modeled as an ellipsoid, of the body Blocking-Body
@ = ^ $ & +           @ body Blocking-Body
- = ^ $ & +         @ intersecting-ray Ray-from-Source-to-Destination
@ = ^ $ & +           @ towards Destination-Point
@ = ^ $ & +           @ from Source-Point
- = ^ $ & +       @ conjunct1 Intersection-is-between-Source-and-Destination?
- = ^ $ & +         @ greater Distance-Source-Destination
@ = ^ $ & +           @ between-point2 Destination-Point
@ = ^ $ & +           @ between-point1 Source-Point
- = ^ $ & +         @ lesser Distance-Source-Intersection
- = ^ $ & +           @ between-point2 Intersection-Point
@ = ^ $ & +             @ intersecting-ellipsoid the surface, modeled as an ellipsoid, of the body Blocking-Body
@ = ^ $ & +             @ intersecting-ray Ray-from-Source-to-Destination
@ = ^ $ & +           @ between-point1 Source-Point
```

Figure 6 Enclosure Blocked Signal in Text Based Interface

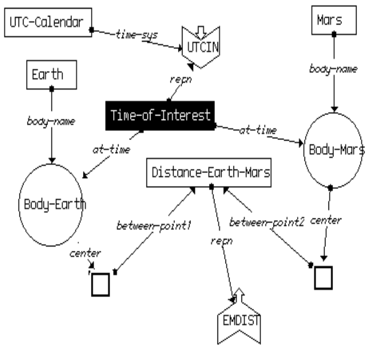
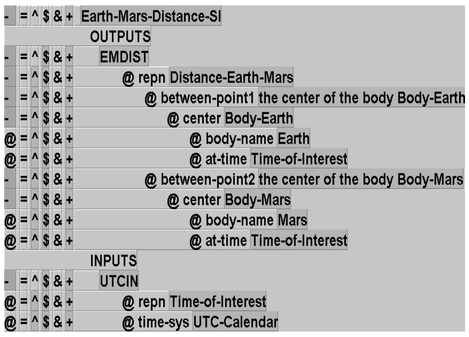


Figure 7 Earth-Mars-Distance graph



```
- = ^ $ & +  Earth-Mars-Distance-SI
             OUTPUTS
- = ^ $ & +    EMDIST
- = ^ $ & +         @ repn Distance-Earth-Mars
- = ^ $ & +              @ between-point1 the center of the body Body-Earth
- = ^ $ & +                   @ center Body-Earth
@ = ^ $ & +                        @ body-name Earth
@ = ^ $ & +                        @ at-time Time-of-Interest
- = ^ $ & +              @ between-point2 the center of the body Body-Mars
- = ^ $ & +                   @ center Body-Mars
@ = ^ $ & +                        @ body-name Mars
@ = ^ $ & +                        @ at-time Time-of-Interest
             INPUTS
- = ^ $ & +    UTCIN
@ = ^ $ & +         @ repn Time-of-Interest
@ = ^ $ & +         @ time-sys UTC-Calendar
```

Figure 8 Earth-Mars-Distance text

7 CONCLUSIONS AND FUTURE WORK

Even though a formal usability study has not yet been conducted, preliminary results indicate that despite the readability advantages provided by the text based interface, the graphical interface also provides advantages of its own, such as a more compact way to present the specifications. Given these observations, we can conclude that it is appropriate to use each interface under different circumstances. The graphical interface is most appropriate when it is important to grasp the high-level structure of the specification. The text based interface is best to provide access to specification details.

An alternative way to present this information is to simultaneously use the text based and the graphical interface to display specification information. Since the new implementation of Amphion is built upon a client server architecture, it is possible to start a graphical and a text based client that communicate through the server so that a given specification can be seen by both clients. Changes made in one of the clients are automatically reflected in the other client. This approach is promising, since it presents the user with the advantages of both representations.

At this point there is a working version of the new Amphion architecture. This new version addresses most of the problems described in Section 1. However, there is still a lot of work to do. Extensive testing of the current prototypes in more realistic settings will take place in the near future.

Future work will focus on extensive testing of the different alternatives to the representation and manipulation of formal specifications described earlier. Since both the text based and the graphical clients will soon be available through the worldwide web, it should be relatively easy to accumulate the necessary data for a formal study.

Current work includes the extension of both of the interfaces to go beyond the creation of specifications into the creation and modification of the domain theory itself. This would allow a domain expert to define new domain theories without having to depend on a programmer to do the domain theory implementation.

Given the constructive theorem proving technique used by Amphion in order to construct the resulting programs, the correctness of these results with respect to the given specification is guaranteed, as long as the domain theory and the system itself are also correct. These properties make it appealing to consider application domains that go beyond software, such as the control of processes in industrial and manufacturing settings. We can visualize Amphion converting an specification describing the safety constraints for an industrial plant into a set of actions that would keep the plant under these constraints throughout a period of time, or creating a daily plan for a production line given the goals of the day.

Research on collaborative construction of formal specifications using Amphion is also being explored. This would allow researchers or students to come together through time and space to build new applications with the assistance of Amphion.

There are several ways we envision the system to be used for educational purposes. First, instructors can use Amphion in order to generate supporting materials for classroom or homework activities, such as animations or tabulations that illustrate particular events. This would be particularly useful for professors teaching elementary courses. Another way to take advantage of these materials is to put together web sites that use those materials in order to present interactive lessons with specific objectives. Advanced students may also use Amphion as a learning environment, by directly interacting with the system in order to obtain answers to problems of their own.

8 REFERENCES

Ladret D. and Rueher M. (1991) VLP: A visual logic programming language. Journal of Visual Languages and Computing, **2**, 2, 163-88.

Lazzeri S. and Heller R. (1996) An Intelligent Consultant System for Chess. Computers & Education, **27**, 3/4, 181-96.

Lowry, M., Philpot, A., Pressburger, T., and Underwood, I. (1994) Amphion: Automatic Programming for Scientific Subroutine Libraries. 8th Intl. Symp. on Methodologies for Intelligent Systems, Charlotte, North. 326-35.

Rugaber, S., Stirewalt, K., and Wills, L. (1996) Understanding Interleaved Code. Automated Software Engineering, **3**, 1/2, 47-76.

Salton, G., Allan, J., and Buckley, C. (1994) Automatic structuring and retrieval of large text files. Communications of the ACM, **37**, 2, 97-108.

Stickel, M., Waldinger, R., Lowry, M., Pressburger T., and Underwood, I. (1994) Deductive Composition of Astronomical Software from Subroutine Libraries. 12th Conference on Automated Deduction, Nancy, France. 341-55.

9 BIOGRAPHY

Dr. Santos G. Lazzeri is a computer scientist for RECOM technologies, NASA Ames Research Center group. He has been collaborating with Ames' Automated Software Engineering group since 1997. Dr. Lazzeri earned D.Sc., and MS degrees in computer science, artificial intelligence, from The George Washington University, (Washington, D.C., U.S.A.). He also has a Licenciatura in Computer Systems Engineering from the Universidad de las Américas, (Puebla, México). His research includes AI, HCI, CBR, Fuzzy Logic, Education, and Computer Chess.